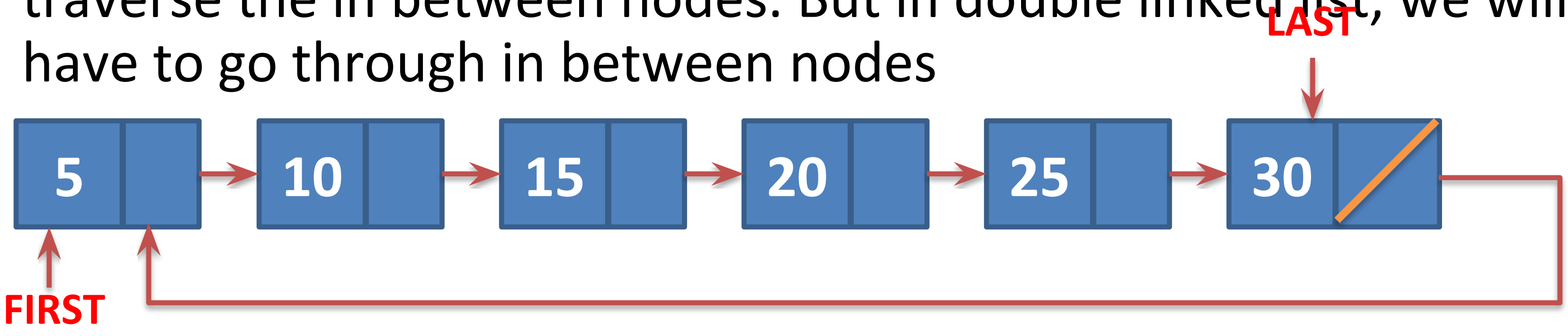
Circular Linked List and Doubly Linked List

Circularly Linked Linear List

- If we **replace NULL** pointer of the **last node** of Singly Linked Linear List with the **address of its first node**, that list becomes circularly linked linear list or **Circular List**.
- **FIRST** is the address of first node of Circular List
- **LAST** is the address of the last node of Circular List
- **Advantages of Circular List**
  - In circular list, every node is accessible from given node
  - It saves time when we have to go to the first node from the last node. It can be done in single step because there is no need to traverse the in between nodes. But in double linked list, we will have to go through in between nodes



Circularly Linked Linear List Cont…

- **Disadvantages of Circular List**
  - It is not easy to reverse the linked list
  - If proper care is not taken, then the problem of infinite loop can occur
  - If we at a node and go back to the previous node, then we can not do it in single step. Instead we have to complete the entire circle by going through the in between nodes and then we will reach the required node

Operations on Circular List

- Insert at First
- Insert at Last
- Insert in Ordered List
- Delete a node

Creation of Circular LinkList

```
void create()
{
        char ch;
        start=NULL;
        do
        {
        new1=(struct node*)malloc(sizeof(struct node));
        printf("\nenter element vlaue:\n");
        scanf("%d",&new1->d);
        if(start==NULL)
        {
                start=new1;
                new1->next=start;
                cur=new1;
        }
        else
        {
                 cur->next=new1;
                 new1->next=start;
                 cur=new1;
        }

                printf("\nenter choice\n");
                ch=getche();
                }while(ch!='n');

                }
```

Display Circular LinkList

```
void display()
{
        ptr=start;
        while((ptr->next)!=start)
        {

                printf("%d-->",ptr->d);
                ptr=ptr->next;

        }
        printf("%d",ptr->d);
}
```

Procedure: CIR_INS_FIRST(X,FIRST)

```
1. [Underflow?]
   IF     AVAIL = NULL
   Then   Write (“Availability
          Stack Underflow”)
          Return(FIRST)

2. [Obtain address of
    next free Node]
    NEW ⍰ AVAIL

3. [Remove free node from
    availability Stack]
    AVAIL ⍰ LINK(AVAIL)

4. [Initialize fields of
   new node]
    INFO(NEW) ⍰ X

5. [Is the list empty?]
    If    FIRST = NULL
    Then LINK(NEW)=NEW
   Return (NEW)
6. [Initialize search for
   a last node]
    SAVE⍰ FIRST
7. [Search for end of list]
  Repeat while LINK (SAVE) ≠ FIRST
   SAVE ⍰ LINK (SAVE)
8. [Set link field of last node
   to NEW]
       LINK(NEW) ⍰ FIRST
   FIRST ⍰ NEW
   LINK (SAVE) ⍰ FIRST
9. [Return first node pointer]
    Return (FIRST)
```

Insertion at beginning of Circular LinkList

```
void insertbegin()
{
struct node *new_node,*a;
new_node=(struct node *)malloc(sizeof(struct node));
printf(" enter element \n");
scanf("%d",&new_node->d);
if(start==NULL)
{
	start=new_node;
	new_node->next=start;
}
else
{
a=start;
while(a->next!=start)
{
	a=a->next;
}
new_node->next=start;
start=new_node;
a->next=start;

}
}
```

Procedure: CIR_INS_END(X,FIRST)

- This procedure **inserts a new node at the last position** of Circular linked list.
- **X** is a new element to be inserted.
- **FIRST** is a **pointer to the first element** of a Circular linked linear list.
- Typical node contains **INFO** and **LINK** fields.
- **NEW** is a temporary pointer variable.

Procedure: CIR_INS_END(X,FIRST)

```
1. [Underflow?]
   IF     AVAIL = NULL
   Then   Write (“Availability
          Stack Underflow”)
          Return(FIRST)

2. [Obtain address of
    next free Node]
    NEW  AVAIL

3. [Remove free node from
    availability Stack]
    AVAIL  LINK(AVAIL)

4. [Initialize fields of
   new node]
    INFO(NEW)  X

5. [Is the list empty?]
     If    FIRST = NULL
     Then LINK(NEW)=NEW
    Return (NEW)
6. [Initialize search for
   a last node]
     SAVE FIRST
7. [Search for end of list]
   Repeat while LINK (SAVE) ≠ FIRST
    SAVE  LINK (SAVE)
8. [Set link field of last node
   to NEW]
    LINK (SAVE)  NEW
    LINK(NEW)  FIRST
9. [Return first node pointer]
     Return (FIRST)
```

Insertion at end of Circular Linked List

```
 void insertend()
{
struct node *new_node,*tmp;
new_node=(struct node *)malloc(sizeof(struct node));
printf(" enter element \n");
scanf("%d",&new_node->d);
if(start==NULL)
{
     start=new_node;
     new_node->next=start;
}
else
{    tmp=start;
       while (tmp->next!=start)
       {
       tmp=tmp->next;
       }
       tmp->next=new_node;
       new_node->next=start;
}
}
```

Deletion Circular Linked List

```
void delbegin()
{
        struct node *ptr;

        ptr=start;
        while(ptr->next!=start)
        {
                ptr=ptr->next;
        }
        start=start->next;
        ptr->next=start;
}
```

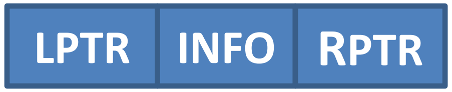
```
void delend()
{
        struct node *b;
        b=start;
        while((b->next)->next!=start)
        {
         b=b->next;
        }
        b->next=start;

}
```

Doubly Linked Linear List

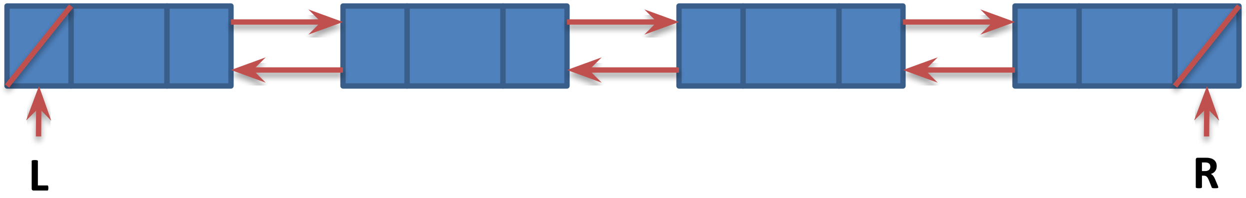
- In certain Applications, it is very desirable that a list be traversed in either forward or reverse direction.
- This property implies that each node must contain two link fields instead of usual one.
- The links are used to denote **Predecessor** and **Successor** of node.
- The link denoting its **predecessor** is called **Left Link.**
- The link denoting its **successor** is called **Right Link.**
- A list containing this type of node is called **doubly linked list** or **two way chain**.

Doubly Linked Linear List

- Typical node of doubly linked linear list contains INFO, LPTR RPTR Fields
- **LPTR** is pointer variable pointing to Predecessor of a node
- **RPTR** is pointer variable pointing to Successor of a node
- Left most node of doubly linked linear list is called **L**, **LPTR** of node L **is always NULL**
- Right most node of doubly linked linear list is called **R**, **RPTR** of node **R is always NULL**



Typical node of Doubly Linked List

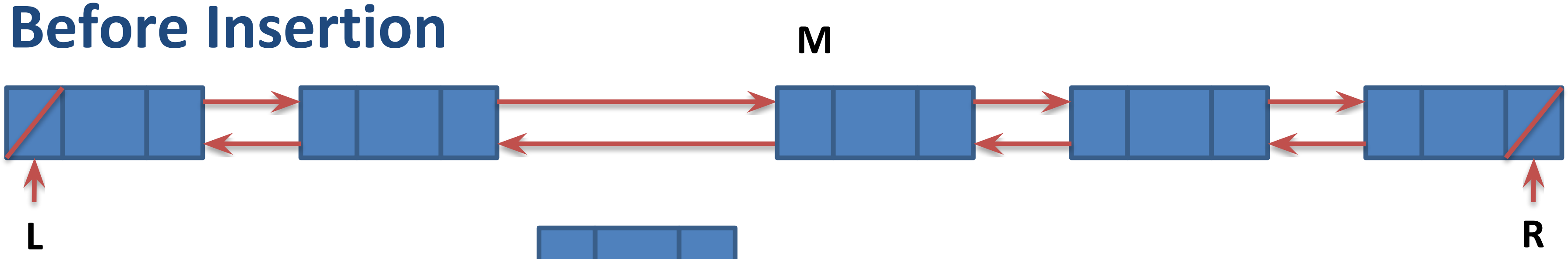


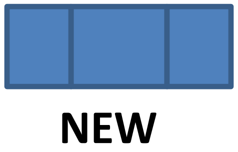
Doubly linked linear list

Insert node in Doubly Linked List

Insertion in the middle of Doubly Linked Linear List

Before Insertion





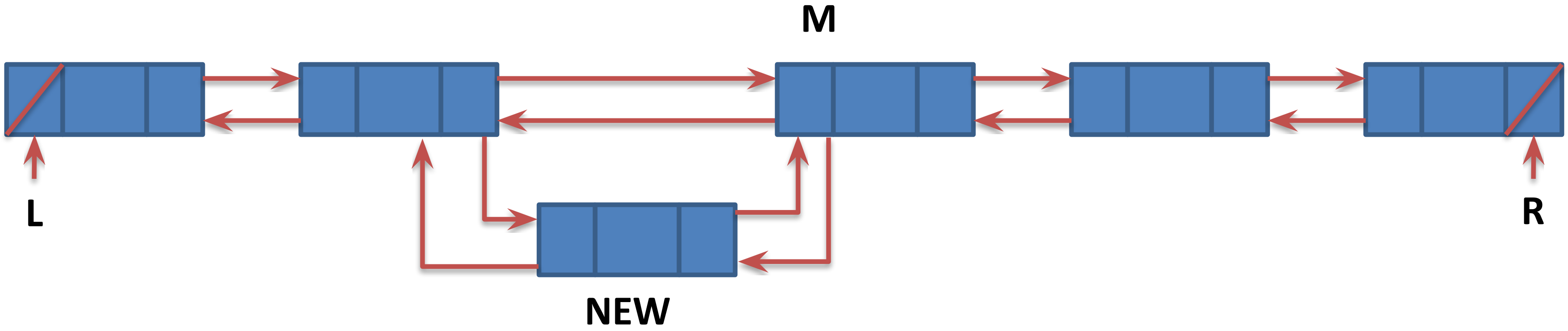
LPTR(NEW) ⍰ LPTR(M)

RPTR(NEW) ⍰ M

LPTR(M) ⍰ NEW

RPTR(LPTR(NEW)) ⍰ NEW

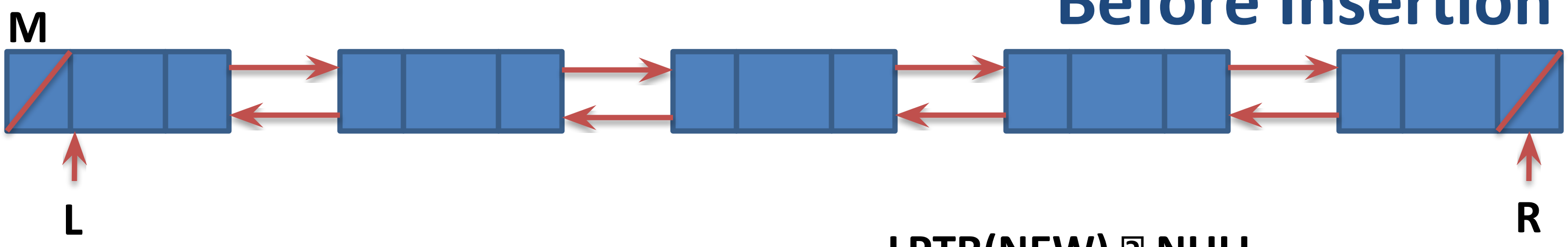
After Insertion

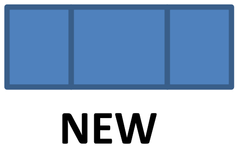


Insert node in Doubly Linked List

Left most insertion in Doubly Linked Linear List

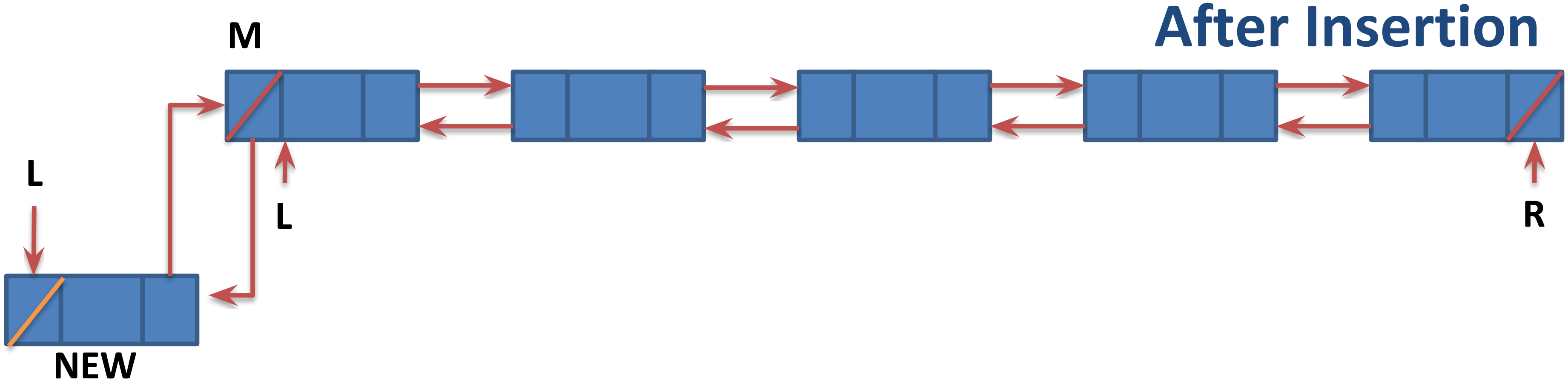
Before Insertion





```
LPTR(NEW) ⍰ NULL
RPTR(NEW) ⍰ M
LPTR(M) ⍰ NEW
L ⍰ NEW
```

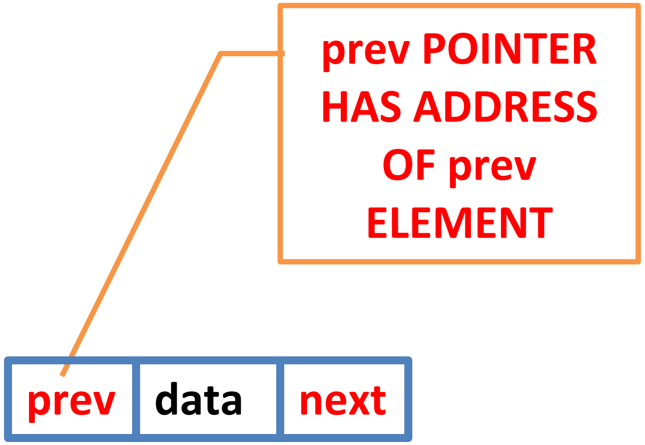
After Insertion



Doubly Linked List

Data Structure for Doubly Linked List:

```
struct node{
	int data;
	struct node *prev;
	struct node *next;
} *start;
```



prev POINTER HAS ADDRESS OF prev ELEMENT

prev	data	next

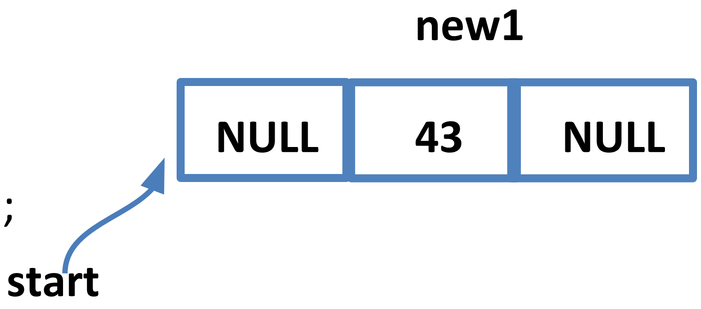
Note :- All the features as it is like singly linked list

Creation of first node in doubly linked list

```
struct node
{
        int d;
        struct node *prev;
        struct node *next;
}*start=NULL, *new1;
```

- new1=(struct node *)malloc(sizeof(struct node *));
- new1->prev=NULL;
- new1->next=NULL;
- printf("\nenter element vlaue:\n");
- scanf("%d",&new1->d);

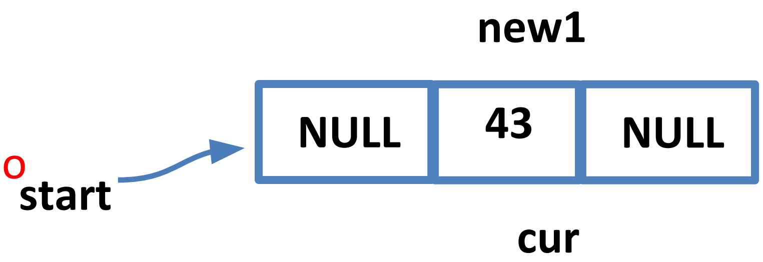
Start=new1;



Creation of doubly linked list

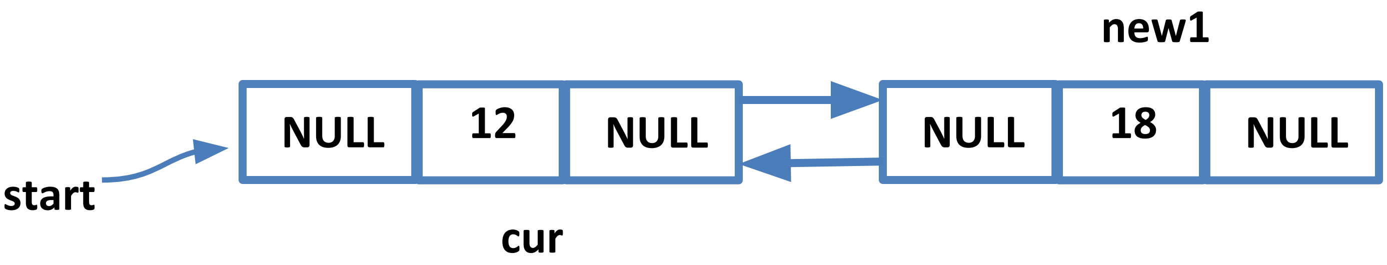
```
void create()
{
        char ch;
        do
        {       new1=(struct node *)malloc(sizeof(struct node *));
                new1->prev=NULL;
                new1->next=NULL;
                printf("\nenter element vlaue:\n");
                scanf("%d",&new1->d);
                if(start==NULL)
                {
                        start=new1;
                        cur=new1;
                }
                else
                {
                    cur->next=new1;
                    cur=new1;
                }
        printf("\nenter choice(press n for exit\n");
        ch=getche();
        }while(ch!='n');
}
```

Initially start=null, so condition is true



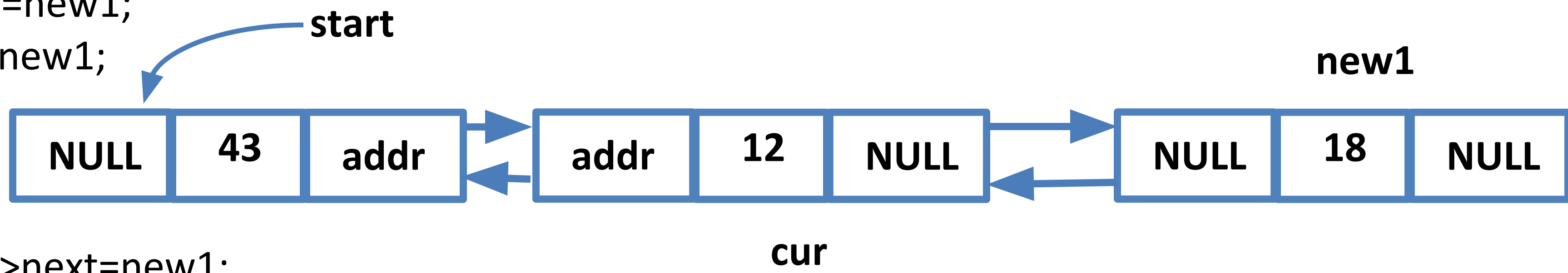
Creation of doubly linked list

```
void create()
{
 char ch;
 do
{   new1=(struct node *)malloc(sizeof(struct node *));
    new1->prev=NULL;
    new1->next=NULL;
    printf("\nenter element vlaue:\n");
    scanf("%d",&new1->d);
    if(start==NULL)  Condition is false
 {
     start=new1;
     cur=new1;
 }
   else
 {
      cur->next=new1;
      new1->pre=cur;
      cur=new1;   }
     printf("\nenter choice(press n for exit\n");
     ch=getche();
}while(ch!='n');
}
```

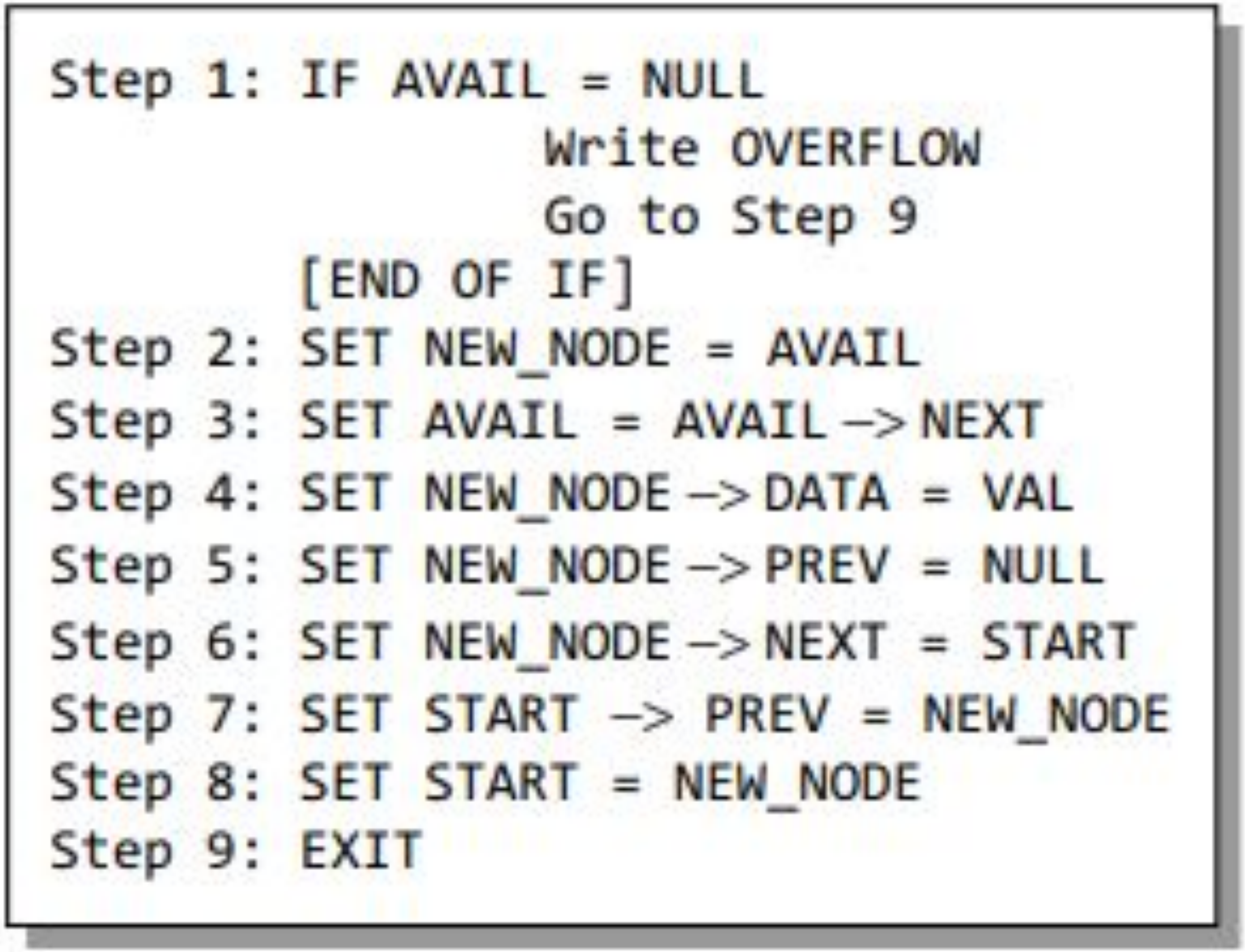


Creation of doubly linked list

```
void create()
{
  char ch;
  do
{   new1=(struct node *)malloc(sizeof(struct node *));
     new1->prev=NULL;
     new1->next=NULL;
     printf("\nenter element vlaue:\n");
     scanf("%d",&new1->d);
     if(start==NULL)  Condition is false
  {
       start=new1;
       cur=new1;
 }
    else
  {
        cur->next=new1;
        new1->pre=cur;
        cur=new1;   }
      printf("\nenter choice(press n for exit\n");
      ch=getche();
}while(ch!='n');
}
```



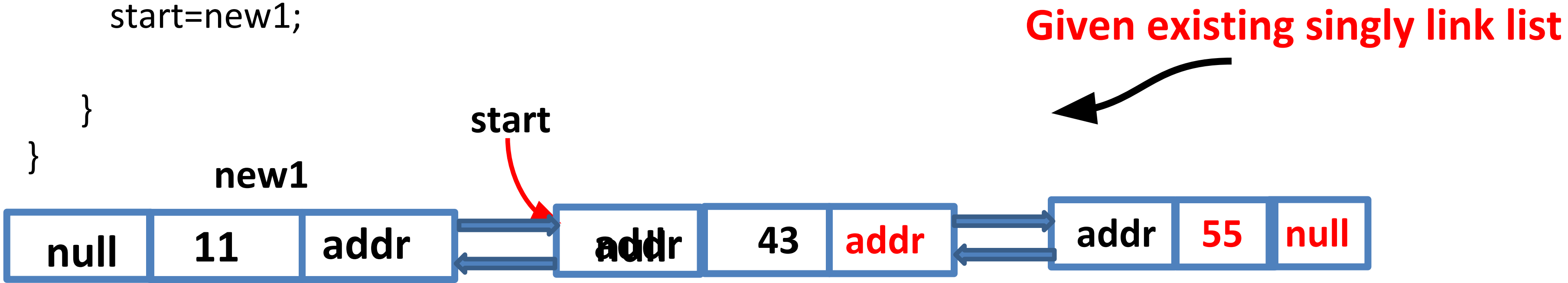
Procedure: DOU_INS (START,X)



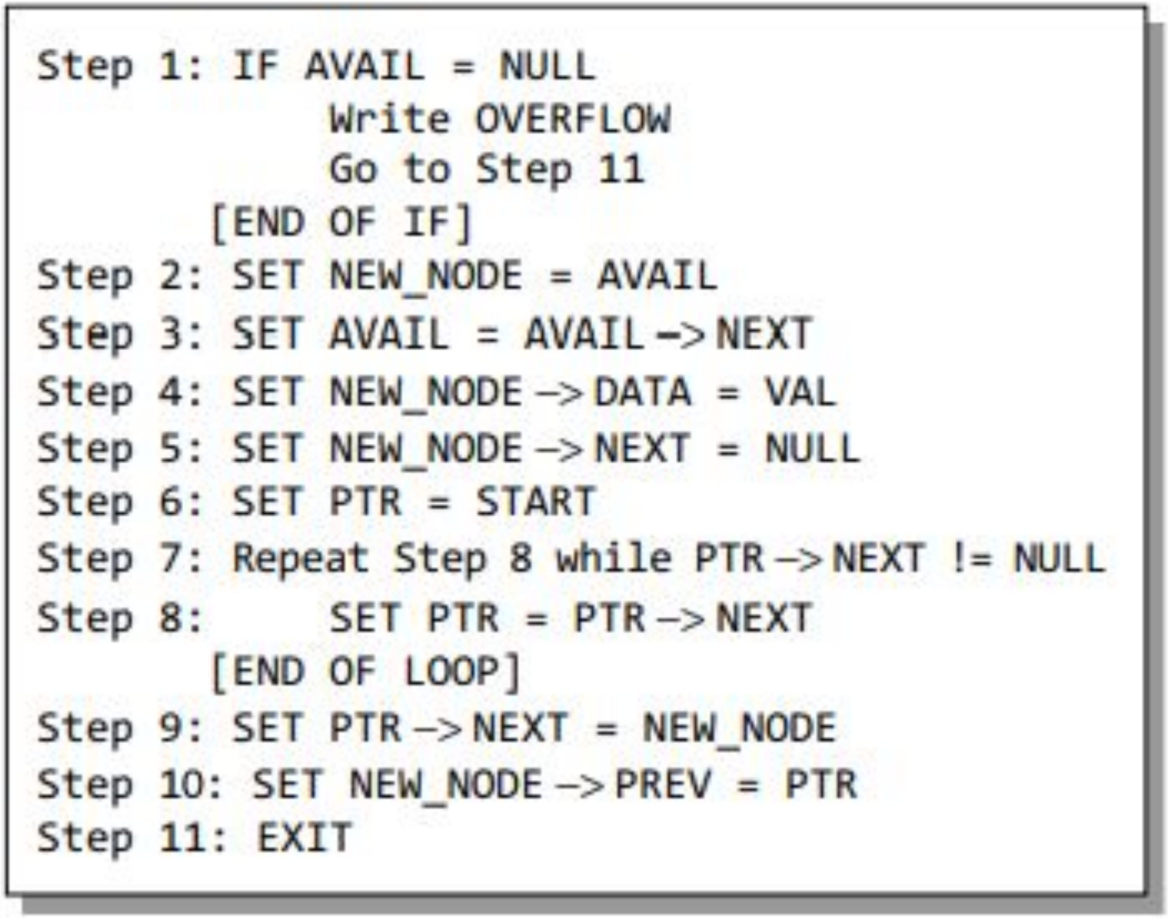
```
Step 1: IF AVAIL = NULL
                Write OVERFLOW
                Go to Step 9
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> PREV = NULL
Step 6: SET NEW_NODE -> NEXT = START
Step 7: SET START -> PREV = NEW_NODE
Step 8: SET START = NEW_NODE
Step 9: EXIT
```

Insert node at beginning of the given doubly linked list

```
void insertbegin()
{
    struct node *new1;
    new1=(struct node *)malloc(sizeof(struct node));
    printf(" enter element \n");
    scanf("%d",&new1->d);
    if(start==NULL)        Condition is false
   {
       start=new1;
       new1->next=NULL;
       new1->prev=NULL;
    }
     else
    {
       new1->pre=NULL;
       new1->next=start;
       start->pre=new1;
       start=new1;

     }
}
```



Procedure: DOU_INS_END (START,X)



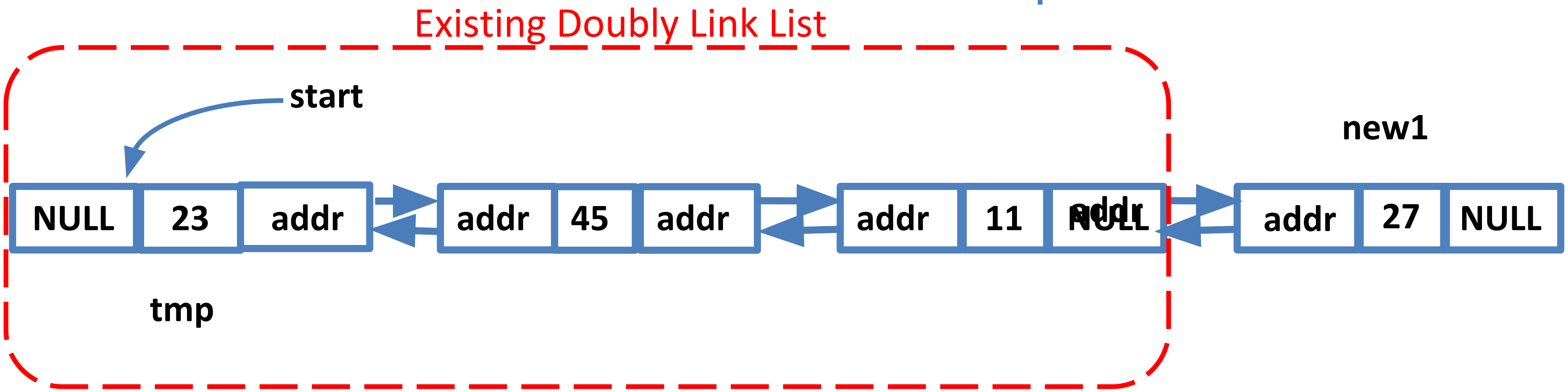
```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 11
       [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = NULL
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR -> NEXT != NULL
Step 8:     SET PTR = PTR -> NEXT
       [END OF LOOP]
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: SET NEW_NODE -> PREV = PTR
Step 11: EXIT
```

Insertion at end of given doubly linked list

```
void insertend()
{
    struct node *new1,*tmp;
    new1=(struct node *)malloc(sizeof(struct node));
    printf(" enter element \n");
    scanf("%d",&new1->d);
      if(start==NULL)
      {
            start=new1;
            new1->next=NULL;
            new1->pre=NULL;
      }
```

Condition is false

```
else {
     tmp=start;
     while (tmp->next!=NULL)
     {
         tmp=tmp->next;
     }
     tmp->next=new1;
     new1->pre=tmp;
     new1->next=NULL;
}
 }
```

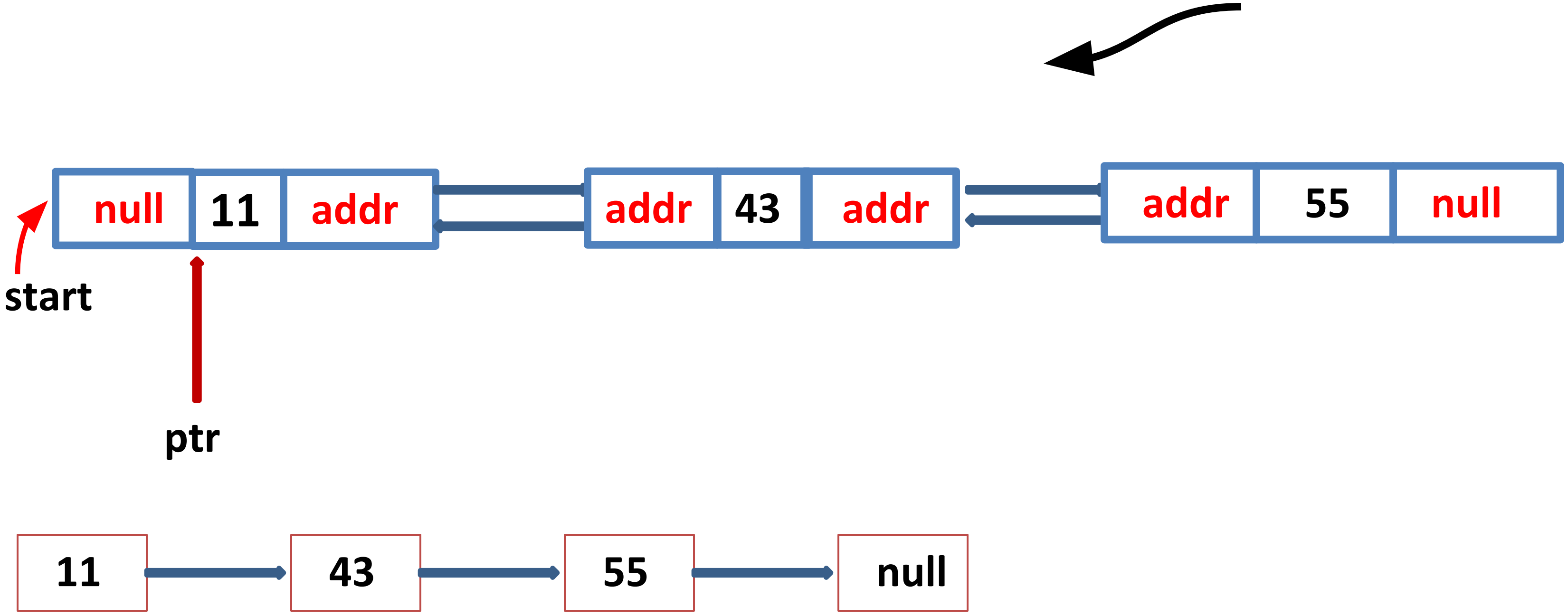


Display given Doubly linked list

```
void display()
{
        ptr=start;
        while((ptr)!=NULL)
        {
                printf("%d-->",ptr->d);
                ptr=ptr->next;
        }
        printf("null\n");
}
```

Condition is false

Given existing singly link list



Insert node after given node in the Doubly linked list

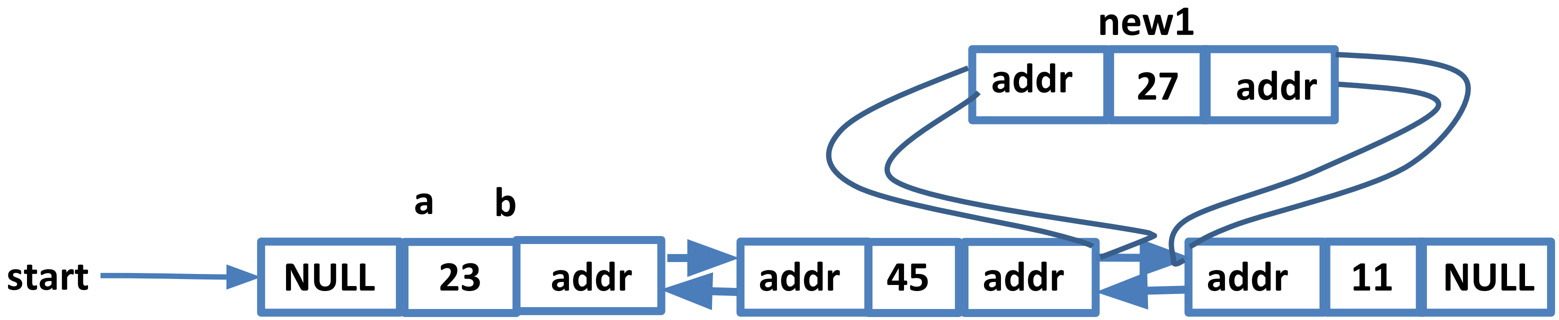
```
void insertafter(){
        int x;
        struct node *new1,*a,*b;
        new1=(struct node *)malloc(sizeof(struct node));
        printf(" enter element \n");
        scanf("%d",&new1->d);
        printf("\nenter a value of a given node after you want to
insert a new node\n");
        scanf("%d",&x);
        b=a=start;
        if(start==NULL){
                printf("empty"); }
        else if(start->d==x){
                b->next=new1;
                new1->pre=b;
                new1->next=NULL;
        }
```

Condition False

```
else{
while(b->d!=x &&b->next!=NULL)
{
        b=a;
        a=a->next;
}
if(b->next==NULL)
{
b->next=new1;
new1->next=NULL;
new1->pre=b;
}
else
{
b->next=new1;
new1->next=a;
new1->pre=b;
a->pre=new1;
}}}
```

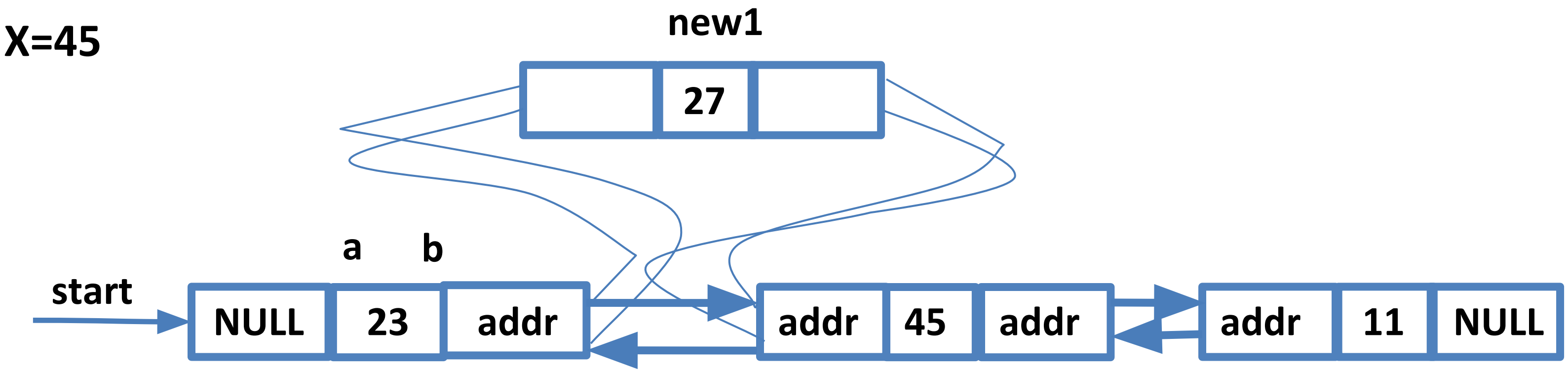
False

X=45



Insert node before given node in the Doubly linked list

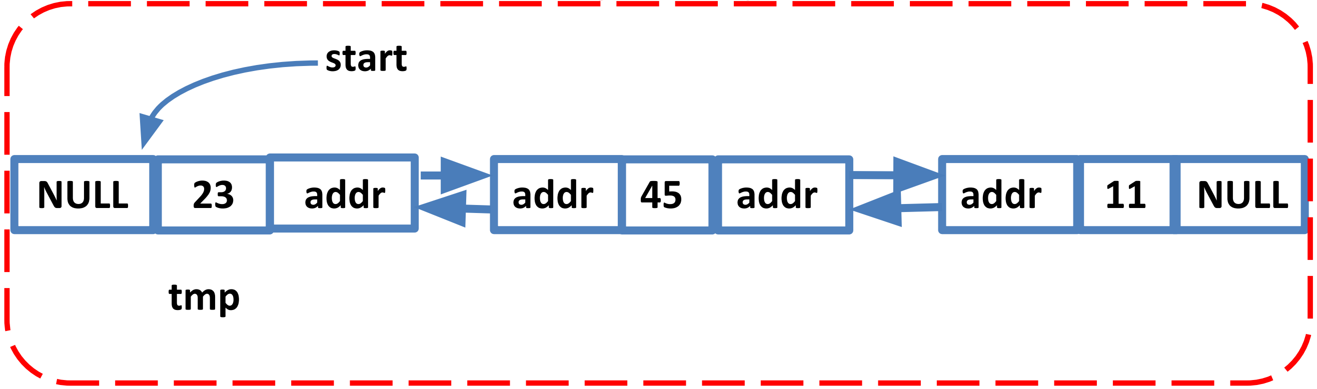
```
void insertbefore(){
        int x;
        struct node *new_node,*a,*b;
        new_node=(struct node *)malloc(sizeof(struct node));
        printf(" enter element \n");
        scanf("%d",&new_node->d);
        printf("enter a value of a given node before you want   to
insert a new node");
        scanf("%d",&x);
        b=a=start;
        if(start==NULL){          false
                printf("empty");}
        else if(start->d==x){          false
           b->pre=new_node;
           new_node->next=b;
           start=new_node;
        }
 else
{
while(a->d!=x)
{
        b=a;
        a=a->next;
}
b->next=new_node;
new_node->next=a;
new_node->pre=b;
a->pre=new_node;
}
}
```



Delete first node from the Doubly linked list

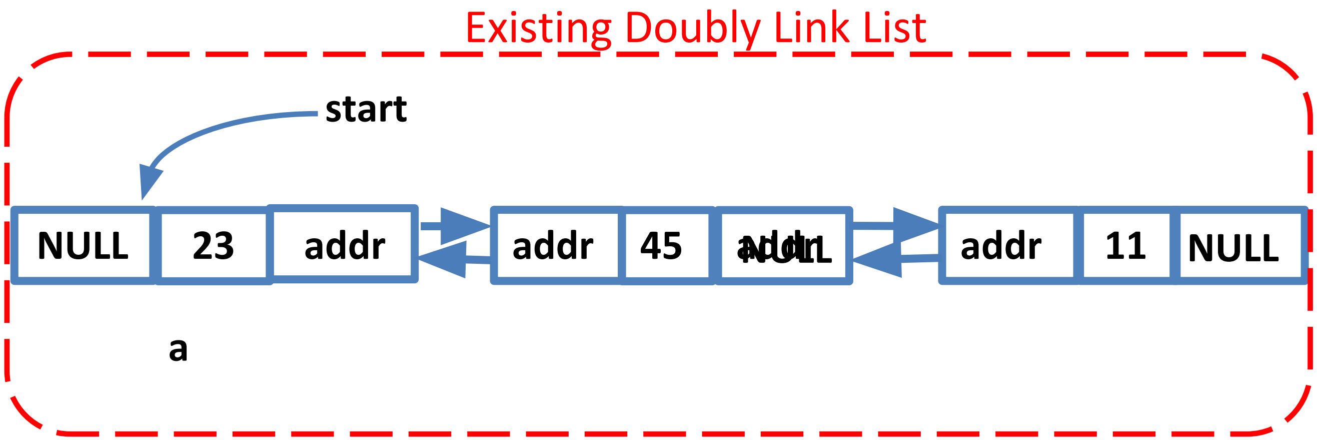
```
void delbegin()
{
        struct node *tmp;
        tmp=start;
        start=tmp->next;
        start->pre=NULL;
        free(tmp);
}
```

Existing Doubly Link List



Delete last node from the singly link list

```
void delend()
{
        struct node *a;
        a=start;

        while(((a->next)->next)!=NULL)
        {
        a=a->next;
        }
        a->next=NULL;

}
```



Delete selected node from the Doubly link list

```
void delselected(){
        struct node *a, *b;
        int x;
        a=b=start;
        printf("\nEnter element to be delete\n");
        scanf("%d",&x);
        while(b->d!=x && b->next!=NULL){
         a=b;
         b=b->next;}
        if(start->next==NULL){
                start=NULL;     }
        else if(b->next==NULL)
        {       if(b->d!=x)
                printf("Element not found");
                else{
                        a->next=NULL;
                        b->pre=NULL;
        }}
        else{
                a->next=b->next;
                (b->next)->pre=a;
        }}
```

Implement Stack Using Link List

```
struct node
{
   int data;
   struct node* next;
};  struct node* top;
void push()
{
        struct node *new_node;
        new_node=(struct node *)malloc(sizeof(struct node));
        printf(" enter element \n");
        scanf("%d",&new_node->d);
        if(top==NULL)
        {
                top=new_node;
                new_node->next=NULL;
        }
        else
        {
                new_node->next=start;
                top=new_node;
        }
}
```

Implement Stack Using Link List

```
void pop()
{
        start=start->next;
}
```

# display() function is same as Singly Link List display().

Implement Queue Using Link List

```
void insert()
{
        struct node *new_node,*tmp,*front,*rear;
        new_node=(struct node *)malloc(sizeof(struct node));
        printf(" enter element \n");
        scanf("%d",&new_node->d);
        new_node->next=NULL;
        if(front==NULL)
        {
                front=rear=new_node;
        }
        else
        {
                rear->next=new_node;
                rear=rear->next;
        }
}
```

Implement Queue Using Link List

```
void delete()
{
    front=front->next;
}
void display()
{
  ptr = front;
  if(front == NULL)
  {
    printf("\nEmpty queue\n");
  }
  else
  {
    while(ptr != NULL)
    {
               printf("\n%d\n",ptr -> data);
               ptr = ptr -> next;
    }
  }
}
```

Applications of linked list

1. Implementation of stacks and queues
2. Implementation of graphs : Adjacency list representation of graphs is most popular which is uses linked list to store adjacent vertices.
3. Dynamic memory allocation : We use linked list of free blocks.
4. Maintaining directory of names
5. Performing arithmetic operations on long integers
6. Manipulation of polynomials by storing constants in the node of linked list
7. representing sparse matrices